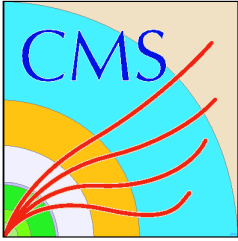


# Scaling HEP to Web Size with RESTful Protocols: The Frontier Example

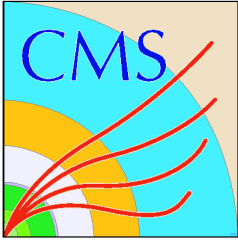
Dave Dykstra, Fermilab  
[dwd@fnal.gov](mailto:dwd@fnal.gov)

Work supported by the U.S. Department of Energy under contract No. DE-AC02-07CH11359



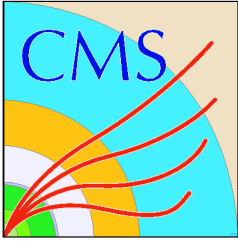
# Introduction

- Talk goal: encourage use of RESTful protocols
- Outline:
  - Explain what REST is about
  - Frontier database caching system as a RESTful protocol example
  - CernVM FileSystem as another example
  - Authorization protocols & REST



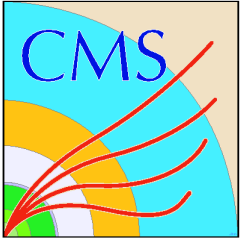
# REST background

- REpresentational State Transfer
- Defined by Roy Fielding in his PhD dissertation
- General architectural style derived from using a subset of http strictly according to http RFCs
  - Roy was a principal author of http RFCs
- Designed for Internet-sized scaling, that is, primarily for **caching**
  - Also for easy replication of servers
- Good for many purposes beyond web browsing
  - including many HEP tasks that need large scaling



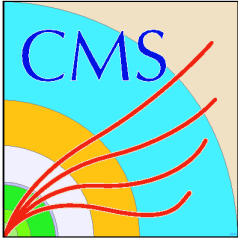
# REST essentials

- Essentials for an http-based cacheable protocol:
  - Stateless service
    - Every request independent
    - No cookies
    - No https (digital signatures for authentication are OK)
      - Unless just for tunneling to scalable private-net server farm
  - Use http methods as originally intended
    - Don't use POST to pass in complex parameters
    - Use GET with a separate URL for every “resource”
  - Set cache expiration times
    - Varies by application



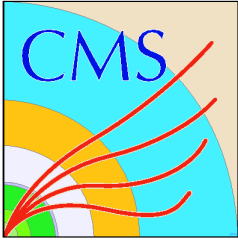
# REST importants

- Important but less essential for an http-based cacheable protocol:
  - Don't use '?' in URL (forces change to cache option)
  - Use Last-Modified/If-Modified-Since or ETag/If-None-Match
    - Enables revalidating cache with simple NOT MODIFIED response if nothing changed
    - If answer has changed, it is returned immediately with no protocol overhead
  - Deploy with sufficient caching proxies



# REST Frontier example

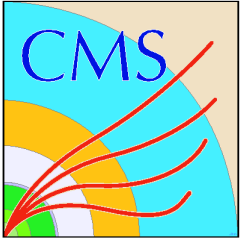
- Distributes read-only database SQL queries
  - Updates are done with a different protocol (like most of the RESTful cacheable systems I have seen)
- Designed for HEP “Conditions” data with many readers of same data distributed worldwide
- Ideal for caching



# REST Data Elements

Data Element    Frontier example

resource	Data returned from an Oracle query
resource identifier	SQL query encoded in URL (by gzip and base64)
representation	XML document including data (encoded by gzip and base64)
representation metadata	Last-Modified time header
resource metadata	Database error messages
control data	If-Modified-Since, Cache-Control



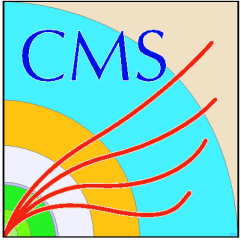
# REST Components

Component    Frontier example

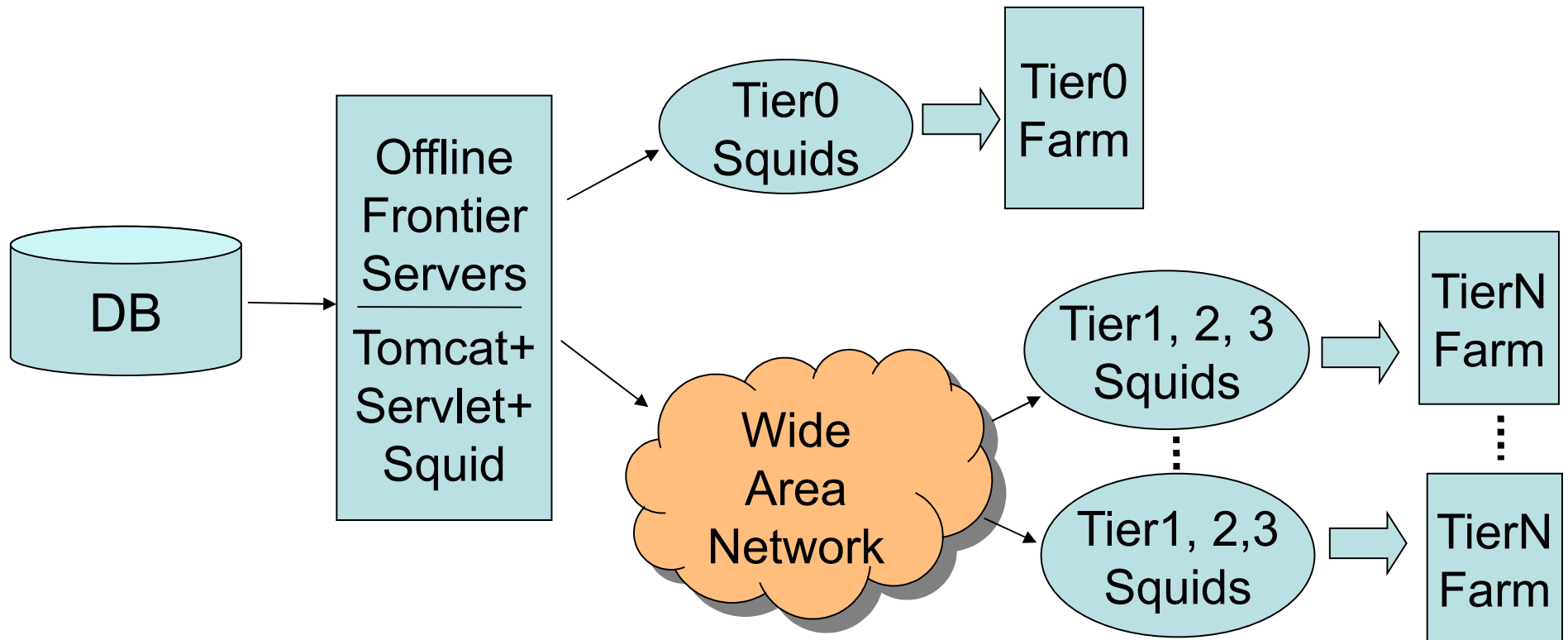
origin server	Tomcat with Frontier servlet
gateway	Squid in “accelerator” or “reverse proxy” mode
proxy	Squid
user agent	frontier_client

“Note that the difference between a proxy and a gateway is that a client determines when it will use a proxy.”

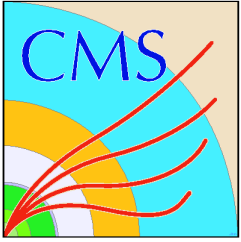
– R. Fielding dissertation



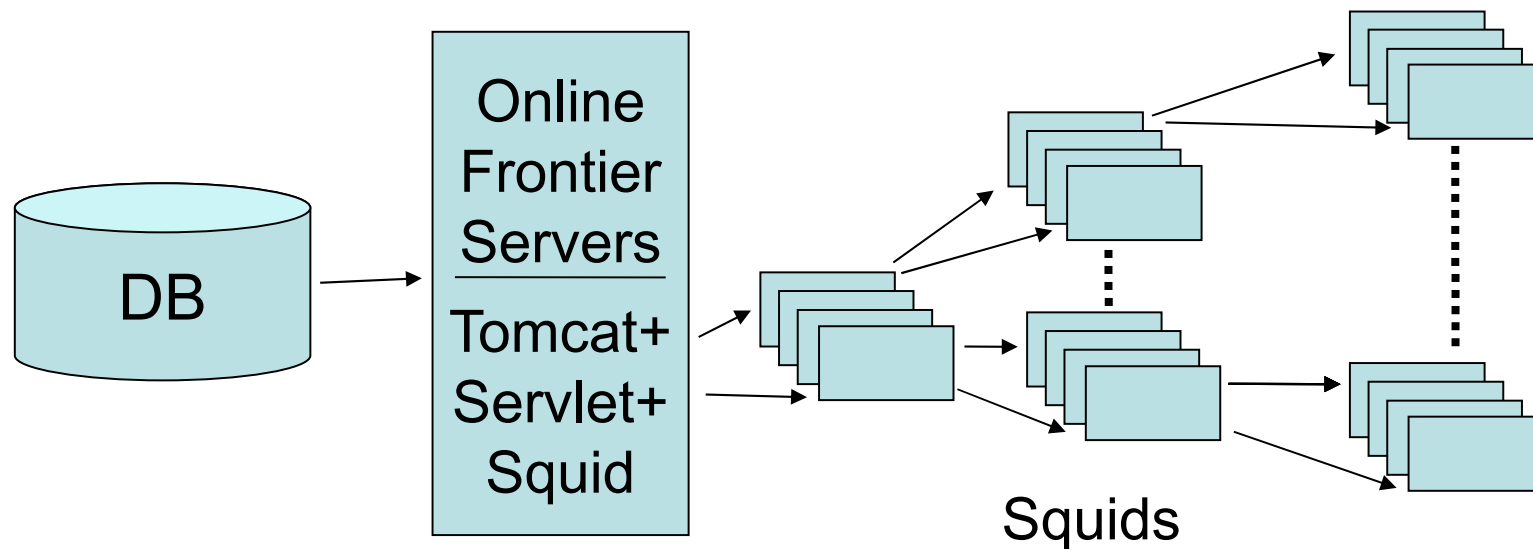
# CMS Offline Frontier example



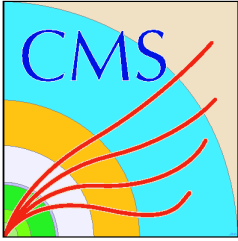
- Many copies of `frontier_client` in jobs on the farms
- Jobs start around the world at many different times
- Cache expirations vary from 5 minutes to a year



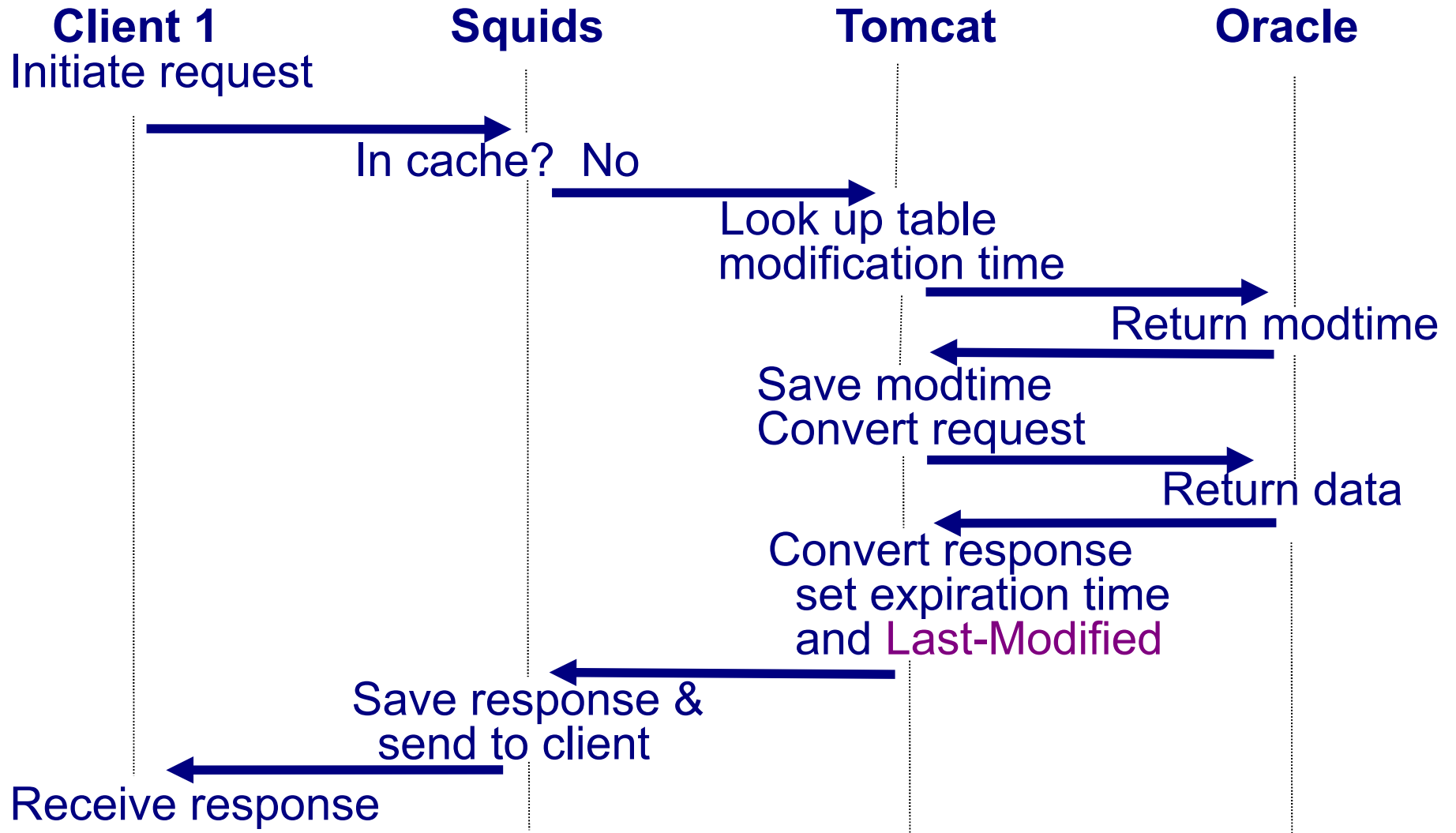
# CMS Online Frontier example

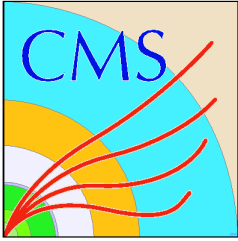


- Blasts data to all 1400 worker nodes in parallel
- Hierarchy of squids on worker nodes
- Frontier servlet sends "Cache-Control: max-age=30"

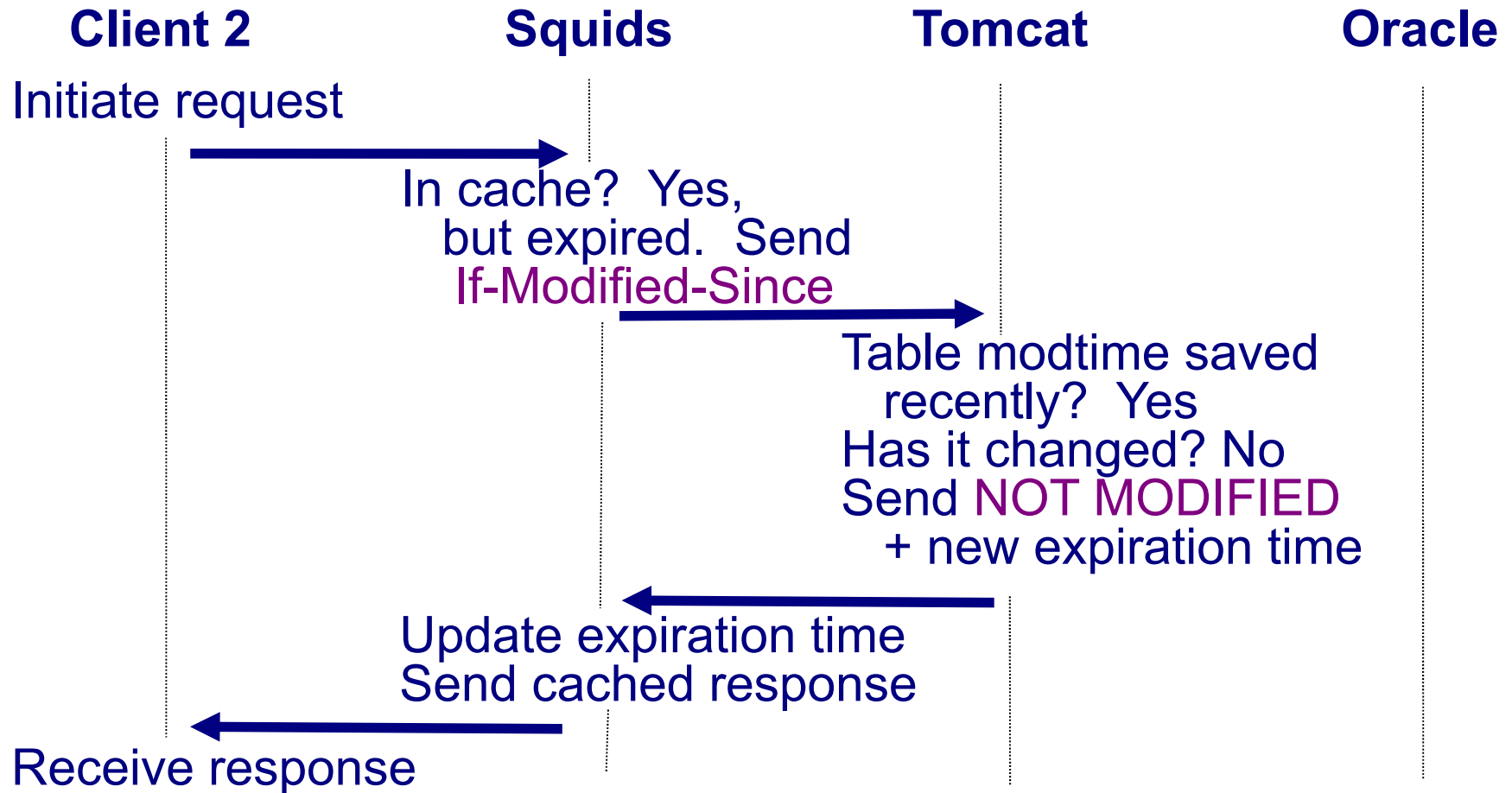


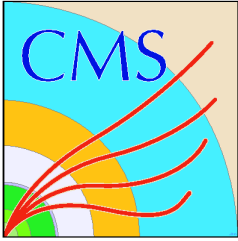
# Filling cache in Frontier





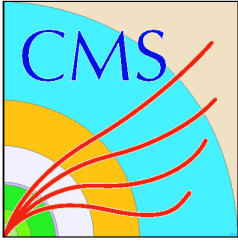
# Validating cache in Frontier





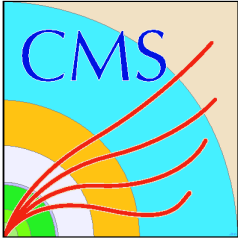
# REST CVMFS example

- CernVM File System (CVMFS) designed to distribute slowly changing filesystem of software
  - Mostly only additions
- URLs are secure hash of **contents** of the files
  - Once cached, they never change
  - Detection of tampering is trivial
- Indexes map filenames to hashes
  - Digitally signed to prevent man-in-the-middle attack



# REST for Authorization

- Grid authorization uses SOAP-based protocols
  - Uncacheable on several levels
    - https
    - POST for parameters
    - Client node name in parameters
  - Many of essentially the same authorization requests happen about the same time so caching could help
    - Sign & timestamp answers instead of encrypt
    - Encode parameters in URL
    - Put a group name in URL instead of node name
- Amazon EC2 has RESTful authorization option



# Summary

- Use REST!
  - Whenever the same information is needed in many places
  - Use locally deployed standard caches
    - Already deployed at most sites participating in LHC experiments